

Logic in Access Control

Martín Abadi

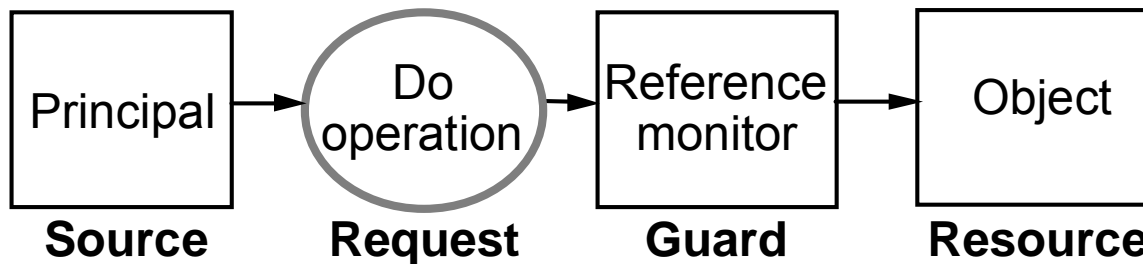
University of California, Santa Cruz

Plan

- Introduction to access control
- Some logical approaches (algorithms, verification, logical languages and theories)
- A logic with “says” for distributed systems
- SDSI
- Binder

The access-control model

- Elements:
 - **Objects** or resources
 - **Requests**
 - Sources for requests, called **principals**
 - A **reference monitor** to decide on requests



Authentication vs. access control

- Access control (authorization):
 - Is principal A trusted on statement S?
 - If A requests S, is S granted?
- Authentication:
 - Who says S?

An access control matrix

[Lampson, 1971]

objects	file1	file2	file3	file4
principals				
user1	rwX	rw	r	X
user2	r	r		X
user3	r	r		X

Implementing access control

Two strategies (often combined):

ACLs and **capabilities**.

- ACL: a column of an access control matrix, attached to an object.
- Capability: (basically) a pair of an object and an operation, for a given principal.
It means that the principal may perform the operation on the object.

Issues

- Access control is pervasive
 - applications
 - virtual machines
 - operating systems
 - firewalls
 - doors
 - ...
- Access control seems difficult to get right.
- Distributed systems make it harder.

General theories and systems

- Over the years, there have been many theories and systems for access control.
 - Logics
 - Languages
 - Infrastructures (e.g., PKIs)
 - Architectures
- They often aim to explain, organize, and unify access control.
- They may be intellectually pleasing.
- They may actually help.

Algorithmic analysis

[starting with Harrison, Ruzzo, and Ullman, 1976]

- A **system** has finite sets of rights and commands.
- A **configuration** is an access control matrix.
- A **command** is of the form “if conditions hold, perform operations” (with some parameters).
 - The conditions are predicates on the matrix.
 - The operations add or delete rights, principals, and objects.
 - E.g.:

```
command CONFER, (owner, friend, file)
  if own in (owner, file)
  then enter r into (friend, file)
end
```

Algorithmic analysis (cont.)

- **Safety** means that untrusted subjects cannot access a resource in any reachable state.
- **Safety is undecidable (in general).**

Algorithmic analysis (cont.)

[in particular, Li, Winsborough, and Mitchell, 2003]

- Not all interesting problems are undecidable!
- Consider the **containment** problem:

In every reachable state, does every principal that has one property (e.g., has access to a resource) also have another property (e.g., being an employee)?

For different classes of systems, this problem is decidable (in coNP or coNEXP).

Formal verification

A formally verified security kernel is widely considered to offer the most promising basis for the construction of truly secure computer systems at least in the short term. A number of kernelized systems have been constructed and various models of security have been formulated to serve as the basis for their verification.

Despite the enthusiasm for this approach there remain certain difficulties and problems in its application [...]

(Rushby, 1981)

A logic from matrices

- An access control matrix may be represented with a ternary predicate symbol **may-access**.
- The setting may be a fairly standard, classical predicate calculus.
- We may then write formulas such as:

may-access(Alice, Foo.txt, Rd)

and rules such as:

may-access(p, o, Wr) \Rightarrow may-access(p, o, Rd)

A logic from matrices: questions

- Does this really help?
 - In describing policies?
 - In analyzing policies?
- We may need many more constructs and axioms for representing security policies.
For example:
 - `may-jointly-access(p,q,o,r)`
 - `owns(p,o)`
 - ...

(When are we done?)

Some references

- “A Unified Framework for Enforcing Multiple Access Control Policies” by Jajodia, Samarati, Subrahmanian, and Bertino (1997)
- “A Logical Language for Expressing Authorizations” by Jajodia, Samarati, and Subrahmanian (1997)
- “A Logical Framework for Reasoning about Access Control Models” by Bertino, Catania, Ferrari, and Perlasca (2003)

Access control in distributed systems

- Many characteristics of distributed systems make access control harder:
 - size
 - faultiness
e.g., revocation messages may get lost
 - heterogeneity
e.g., of communication channels
of protection mechanisms
 - autonomy and lack of central administration
and therefore of central trust
 - ...

An approach

- A notation for representing principals and their statements, and perhaps more:
 - objects and operations,
 - trust,
 - channels,
 - ...
- Derivation rules

Some early ideas

(Excerpts of a message from Roger Needham, August 1987)

- Notations

$P \models S$ Principals in the set P are
the guarantors for S

$C \Vdash S$ Channel C actually asserts S

$C \rightarrow P$ Channel C authenticates Principal P

- Postulates

$C \Vdash S, C \rightarrow P$

$P \models S$

A calculus for access control

[Abadi, Burrows, Lampson, and Plotkin, 1993]

- A simple notation for assertions
 - A says S
 - A speaks for B
- With logical rules
 - $\vdash A \text{ says } (S \Rightarrow T) \Rightarrow (A \text{ says } S) \Rightarrow (A \text{ says } T)$
 - If $\vdash S$ then $\vdash A \text{ says } S$.
 - $\vdash A \text{ speaks for } B \Rightarrow (A \text{ says } S) \Rightarrow (B \text{ says } S)$
 - $\vdash A \text{ speaks for } A$
 - $\vdash (A \text{ speaks for } B \wedge B \text{ speaks for } C) \Rightarrow A \text{ speaks for } C$

An example

- Let good-to-delete-file1 be a proposition.
Let **B controls S** stand for
 $(B \text{ says } S) \Rightarrow S$
- Assume that
 - B controls (A speaks for B)
 - B controls good-to-delete-file1
 - B says (A speaks for B)
 - A says good-to-delete-file1
- We can derive:
 - B says good-to-delete-file1
 - **good-to-delete-file1**

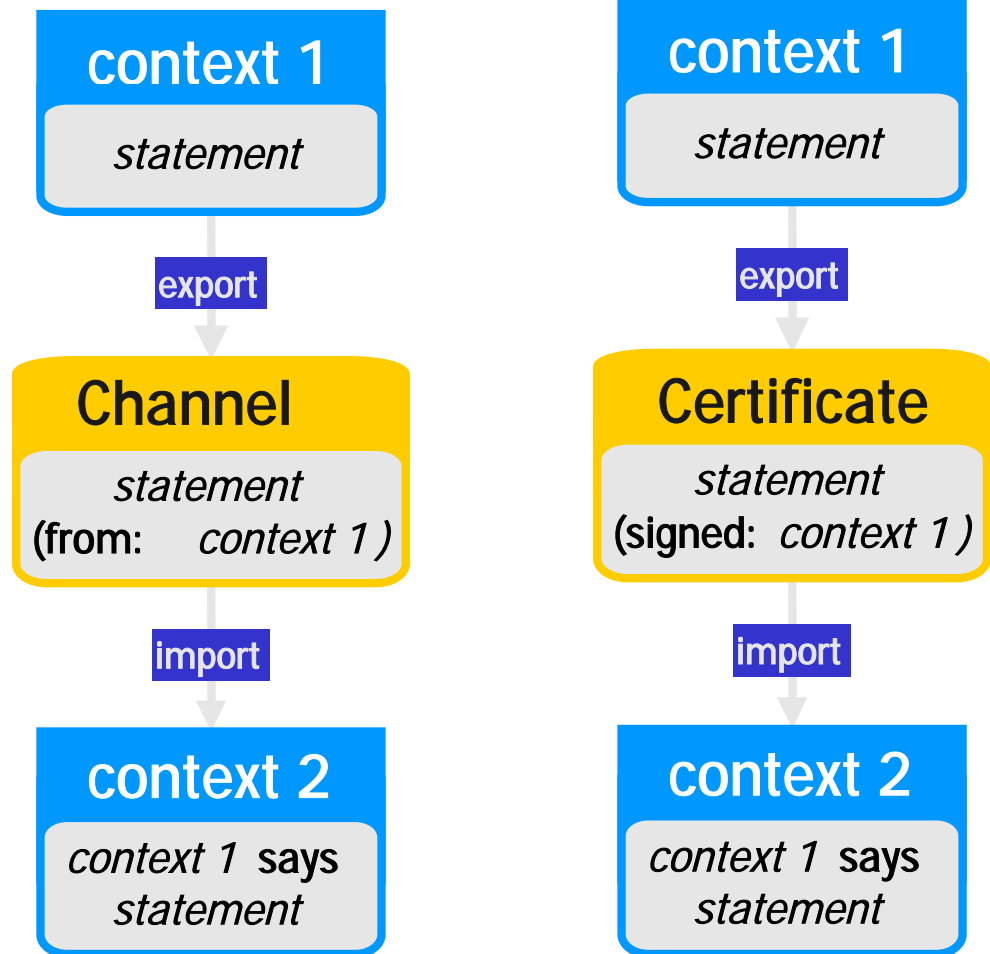
Another example

- Let good-to-delete-file2 be a proposition too.
- Assume that
 - B controls (A speaks for B)
 - B controls good-to-delete-file1
 - B says (A speaks for B)
 - A says (good-to-delete-file1 \wedge good-to-delete-file2)
- We can derive:
 - B says good-to-delete-file1
 - good-to-delete-file1

Says

Says represents communication across contexts.

Says abstracts from the details of authentication.



Choosing axioms

- Standard modal logic?
 - (As above.)
- Less?
 - Treat says “syntactically”, with no special rules (Halpern and van der Meyden, 2001)
- More?
 - $\vdash S \Rightarrow (\text{A says } S)$
(Lampson, 198?; Appel and Felten, 1999)
but then $\vdash (\text{A says } S) \wedge \neg S \Rightarrow (\text{A says false})$

Semantics

- Following standard semantics of modal logics, a principal may be mapped to a binary relation on possible worlds.

A says S holds at world w
iff
S holds at world w'
for every w' such that w A w'

- This is formally viable, also for richer logics.
- It does not give any insight on the nature of authority and duty, but it is sometimes useful.

Proof strategies

- Style of proofs:
 - Hilbert systems
 - Tableaux
(Massacci, 1997)
 - ...
- Proof distribution:
 - Proofs done at reference monitors
 - Partial proofs provided by clients
(Wobber et al., 1994; Appel and Felten, 1999)
 - With certificates pulled or pushed

Compound principals

- Compound principals represent a richer class of sources for requests:
 - $A \wedge B$ Alice and Bob
(cosigning a document)
 - $A \text{ for } B$ ws17.uxyz.edu for Alice
(connecting to the web server)
 - $A \text{ quoting } B$ server.uxyz.edu quoting Alice
(sending a file to the printer)
 - $A \text{ as } R$ Alice as Reviewer
(opening submissions)
- $A \wedge B$ speaks for A, etc.

Groups

- Groups can be seen as disjunctions.
- So we may represent each group by a principal. Then, when A is a member of G , we may write that A speaks for G .
- In practice, it is harder to know when A is *not* a member of G .

Running programs

- Programs need not be fully trusted, not even the operating system.
- Programs can be put into groups.
- Formally, programs can be treated much like roles, and even as roles.

An example

- The cast:
 - CA, the certification authority, with public key K_{CA}
 - WS, a workstation, with public key K_{WS}
 - OS, an operating system, with no key
 - (WS as OS), the resulting node, with ephemeral public key K_n
 - bwl, a user, with public key K_{bwl}
 - K_{del} , an ephemeral public key for the node for bwl
 - C, a secure channel to a file server
 - TrustedNode and SysAdm, two groups

An example (cont.)

- K_{CA} says (K_{WS} speaks for WS)
- K_{WS} says (K_n speaks for (WS as OS))
- K_{CA} says (K_{bwl} speaks for bwl)
- K_{bwl} says (K_{del} speaks for ((WS as OS) for bwl))
- K_n says (K_{del} speaks for ((WS as OS) for bwl))
- K_{del} says (C speaks for ((WS as OS) for bwl))
- C says good-to-delete-file1
- And we may deduce:
((WS as OS) for bwl) says good-to-delete-file1

An example (cont.)

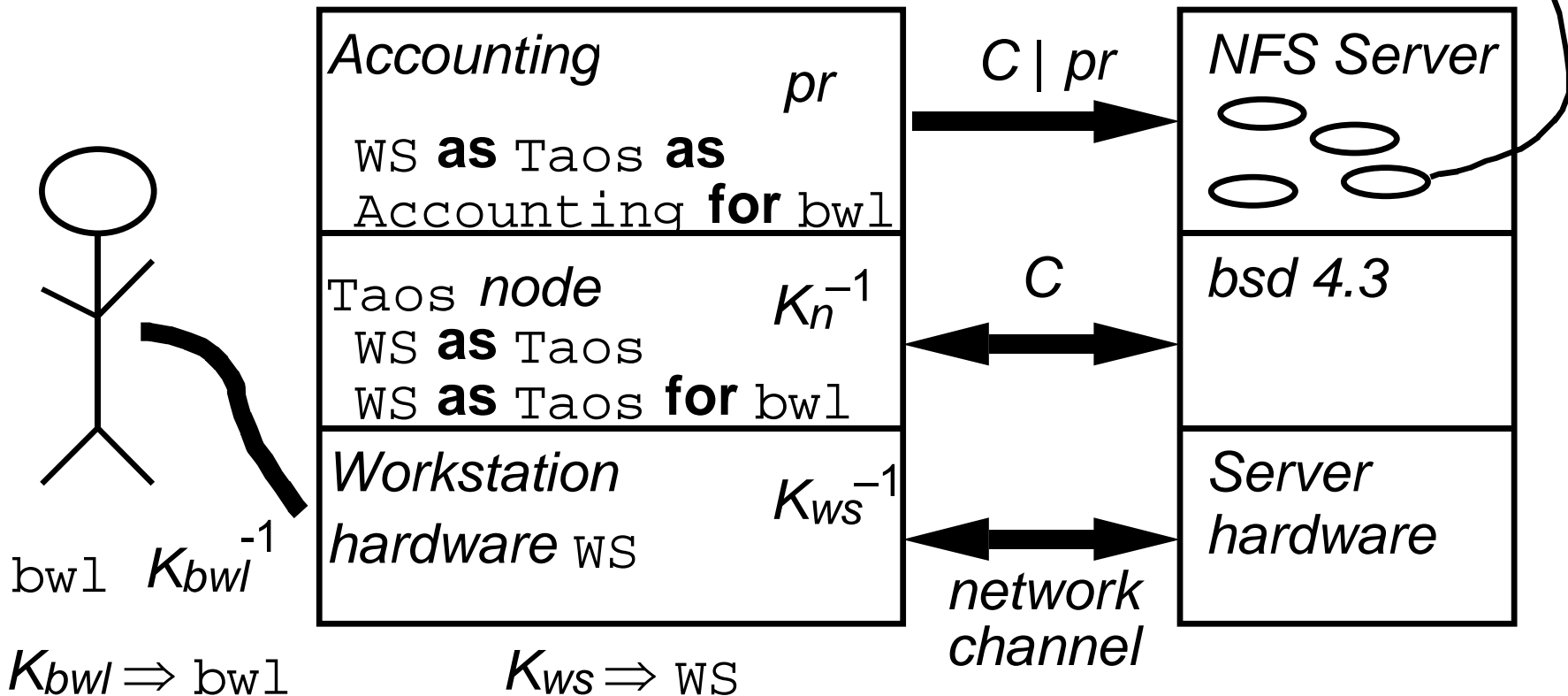
- K_{CA} says ((WS as OS) speaks for TrustedNode)
- K_{CA} says (bwl speaks for SysAdm)
- Then we may deduce:
TrustedNode for SysAdm
says good-to-delete-file1
- The ACL for file1 may say:
TrustedNode for SysAdm
controls good-to-delete-file1
- Then we conclude: good-to-delete-file1

Applications (1): Security in an operating system [Wobber et al., 1994]

SRC-node **as** Accounting **for** bwl
may read

file foo

WS **as** Taos \Rightarrow SRC-node



Applications (2): An account of security in JVMs [Wallach and Felten, 1998]

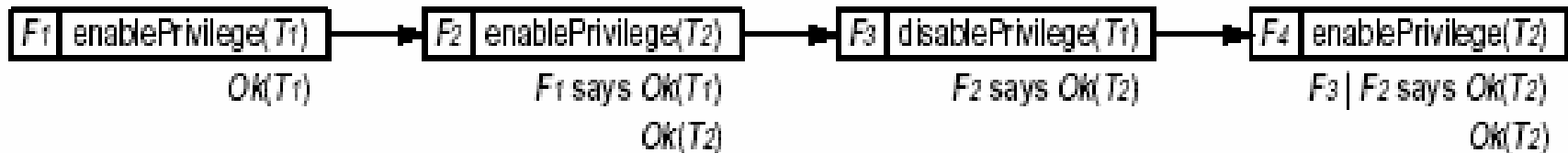
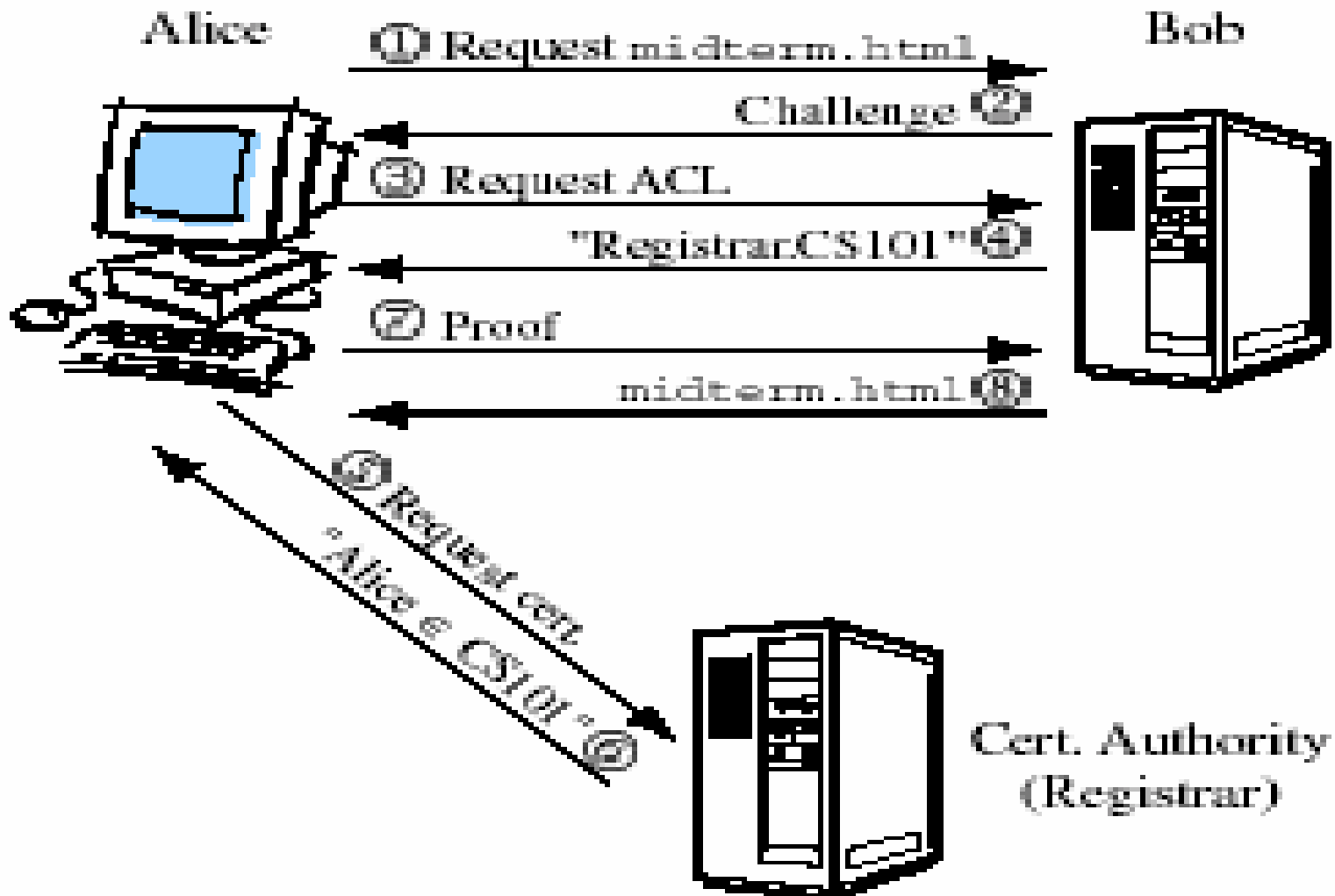


Figure 2: Example of interaction between stack frames. Each rectangle represents a stack frame. Each stack frame is labeled with its name. In this example, each stack frame makes one `enablePrivilege()` or `disablePrivilege()` call, which is also written inside the rectangle. Below each frame is written its belief set after its call to `enablePrivilege()` or `disablePrivilege()`.

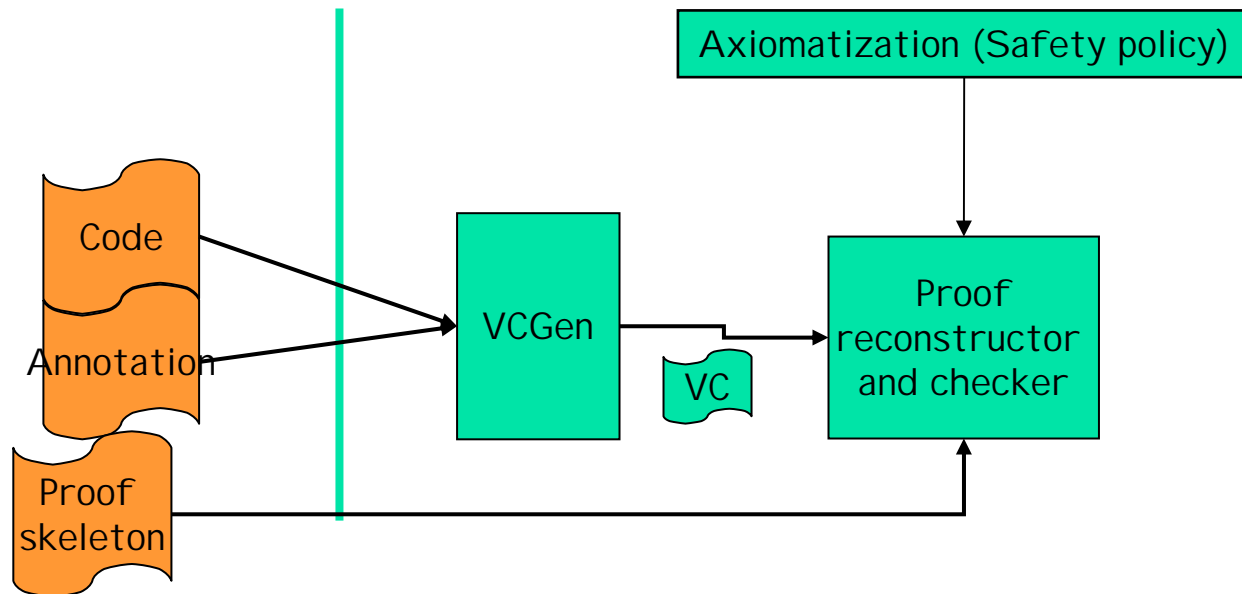
Applications (3): A Web access control system [Bauer, Schneider, and Felten, 2002]



Proof-carrying code (PCC)

[Necula and Lee, 1996]

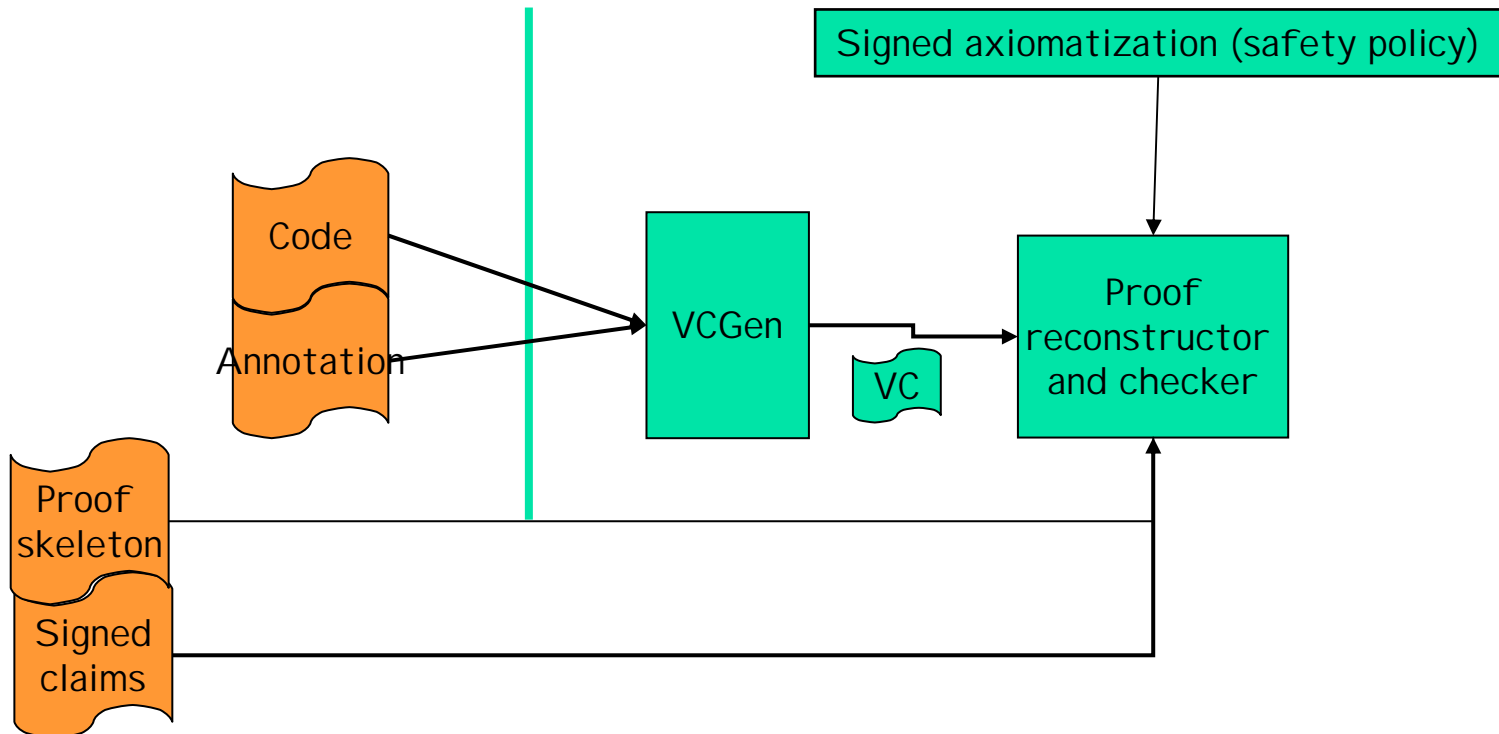
- Proof-carrying code is also based on logic.
- It is also essentially concerned with an authorization decision (running code):



Proof-carrying code (cont.)

[ongoing work with Necula and Whitehead]

- How does PCC fit into the broader context of access control?
- How about hybrid policies and mechanisms?



Other languages and systems

- PolicyMaker and KeyNote [Blaze et al.]
- SDSI [Lampson and Rivest]
- SPKI [Ellison et al.]
- D1LP and RT [Li et al.]
- SD3 [Jim]
- Binder [DeTreville]
- XrML 2.0
- ...

- Several of the most recent are based on ideas and techniques from logic programming.

SDSI (a Simple Distributed Systems Security Infrastructure)

- SDSI includes support for hierarchical public-key certification (in the style of X.509).
 - There can be a tree of authorities for associating principals with their public keys.
- It also supports local name spaces.
 - Each principal can bind local names at wish.
- Local name spaces can be linked.
 - A local name space may bind bwl to Ron's Butler.
- SDSI allows definitions of groups and ACLs.
- SDSI was merged into SPKI, a Simple Public-Key Infrastructure.

The meaning(s) of bindings

- In SDSI, “principals are public keys”.
- So, what is the meaning of bindings?

lawyer \mapsto K means that:

lawyer is K

or that

K speaks for lawyer

(or some other asymmetric relation)

bwl \mapsto Ron's Butler means that:

bwl is Ron's Butler

or that

Ron's Butler speaks for bwl ??

The meaning(s) of bindings (cont.)

- Only asymmetric meanings are viable.
- In fact, “speaks for” (or something similar) was intended by the authors.
- There are further delicate points and ambiguities, e.g., early vs. late binding.

An algorithm for name resolution

Ref2(o, p) = if p is a global identifier f
then return f
else if p is a local name n
and $n \mapsto q$ in assumptions(o)
then return Ref2(o, q)
else if p is a compound name q's r
then return Ref2(Ref2(o, q), r)
else fail

Some axioms [Abadi, 1998]

- Reflexivity $p \mapsto p$
 - Transitivity $p \mapsto q \Rightarrow (q \mapsto r \Rightarrow p \mapsto r)$
 - Left-mono $p \mapsto q \Rightarrow (p's r \mapsto q's r)$
 - Globality $p's g \mapsto g$ if g is a global name
 - Associativity $(p's q)'s r \mapsto p's (q's r)$
 $p's (q's r) \mapsto (p's q)'s r$
 - Linking $p \text{ says } (n \mapsto r) \Rightarrow (p's n \mapsto p's r)$
 - Speaking-for $p \mapsto q \Rightarrow (q \text{ says } S \Rightarrow p \text{ says } S)$
- + standard propositional logic
- + standard modal logic for says

Some semantics

(Again, borrow from modal logics.)

Some results and further work

- The algorithm can be simulated in the logic:
given a set of assumptions E , if $\text{Ref2}(o, p) = g$
then $E \Rightarrow (o\text{'s } p \mapsto g)$ is provable.
- This logic is strictly stronger than the algorithm
(in several ways).
- SDSI and SPKI have rough corners anyway.
- There has been substantial subsequent work on
logical accounts of SDSI and related systems.
 - With special logics
(e.g., Halpern and van der Meyden, 2001).
 - With first-order logic
(Li and Mitchell, 2003).

Alternative axioms

[Halpern and van der Meyden, 2001]

- Reflexivity $p \mapsto p$
- Transitivity $p \mapsto q \Rightarrow (q \mapsto r \Rightarrow p \mapsto r)$
- Left-mono $p \mapsto q \Rightarrow (p's r \mapsto q's r)$
- Globality $K's g \mapsto g$ if g is a global name
and K is a key
- (Two more globality axioms)
- Associativity $(p's q)'s r \mapsto p's (q's r)$
 $p's (q's r) \mapsto (p's q)'s r$

Alternative axioms (cont.)

- Linking $K \text{ cert } (n \mapsto r) \Rightarrow (K\text{'s } n \mapsto p\text{'s } r)$
 - Key distinctness $\neg(K \mapsto K_1)$ for K and K_1
distinct keys
 - Non-emptiness $(p \mapsto K_1) \Rightarrow (p\text{'s } K \mapsto K)$
 - (Three more non-emptiness axioms)
- + standard propositional logic
- + additional axioms if the set of keys is finite
- but nothing more for cert!

Alternative axioms (cont.)

- Closer fit with Ref2
- Closer fit with a semantics
- More complexity
- Different account of “saying”

Comments

We remark that, in a sense, our task is much easier than Abadi's, since we give the constructs in the logic a somewhat narrower reading than he does. Abadi tends to intertwine and occasionally identify issues of naming and issues of rights and delegation. (Such an identification is also implicitly made to some extent in designs such as PolicyMaker [BFL96].) We believe that it is important to treat these issues separately.

(Halpern and van der Meyden)

Binder

- Binder is a relative of Prolog.
- Like Datalog, it lacks function symbols.
- It also includes the special construct says.
- It does not include much else.

An example in Binder

- Facts
 - `owns(Alice, Foo.txt).`
 - `Alice says good(Bob).`
- Rules
 - `may_access(p, o) :- owns(q, o), blesses(q, p).`
 - `blesses(Alice, p) :- Alice says good(p).`
- Conclusions
 - `may_access(Bob, Foo.txt).`

Binder's proof rules

- Binder includes a standard resolution rule.
- In addition, Binder includes a rule for importing formulas from a context F to a context D .
 - The rule adds a “ F says” in front of all atoms without a “says”.
 - The rule applies only to clauses where the head atom does not have “says”.

Binder's proof rules: example

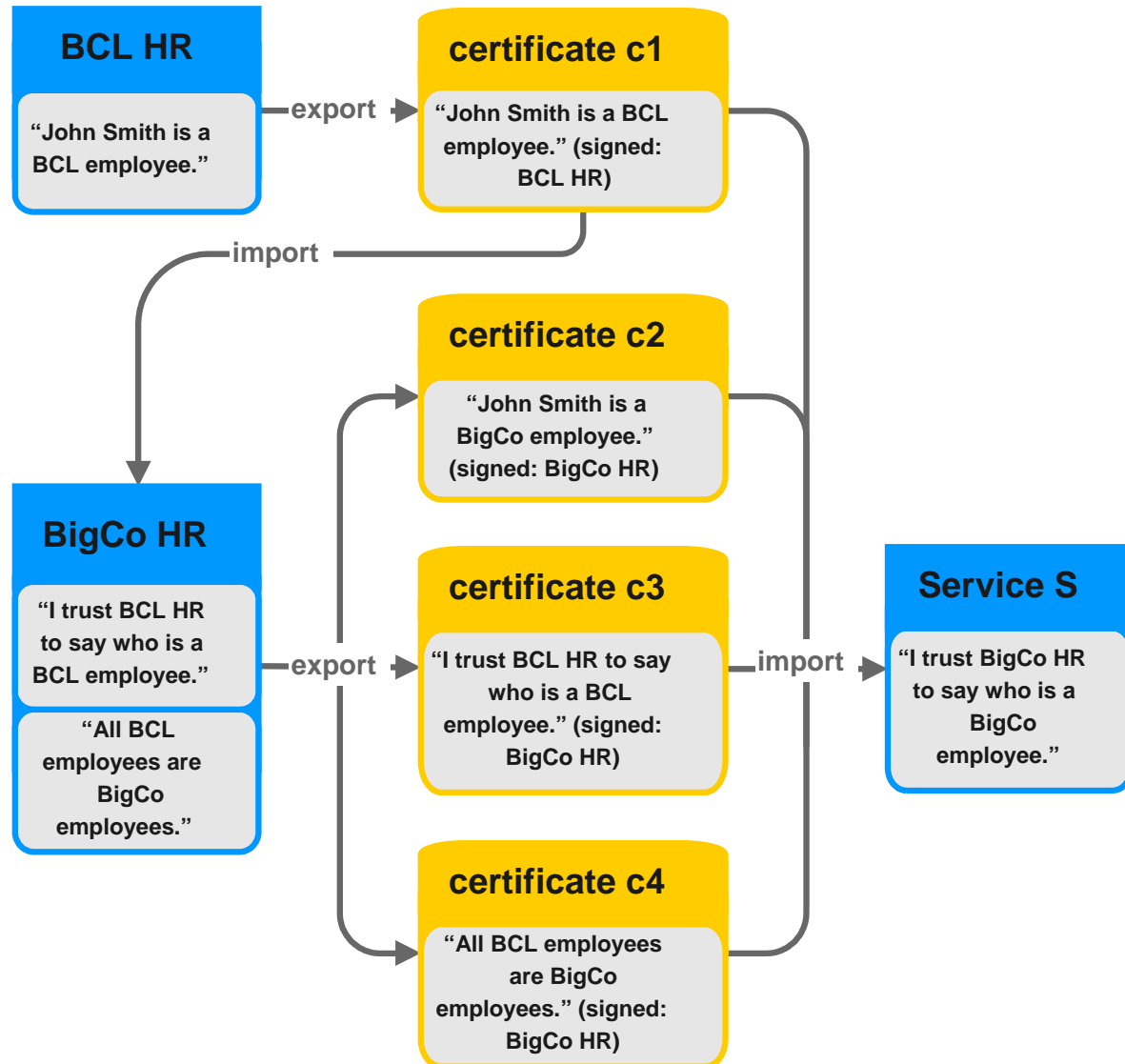
- Suppose F has the rules
 - $\text{may_access}(p, o) \text{ :- owns}(q, o), \text{blesses}(q, p).$
 - $\text{blesses}(\text{Alice}, p) \text{ :- Alice says good}(p).$
 - $\text{Alice says good}(\text{Bob}).$
- D may import the first two as:
 - $\text{F says may_access}(p, o) \text{ :-}$
 $\text{F says owns}(q, o), \text{F says blesses}(q, p).$
 - $\text{F says blesses}(\text{Alice}, p) \text{ :- Alice says good}(p).$
- D may import $\text{good}(\text{Bob})$ directly from Alice.

Binder's proof rules (cont.)

- Suppose F has the rule
 - `blessees(Alice, p) :- Alice says good(p).`
- D may import it as:
 - `F says blessees(Alice, p) :- Alice says good(p).`
- D and F should agree on Alice's identity.
- But the meaning of predicates may vary, and it typically will.
For example, F may also have:
 - `blessees(Bob, p) :- Bob says excellent(p).`

Another example

[DeTreville]



A logical analysis

- Suppose that A has the rule:
 $p \text{ :- } B \text{ says } q, r$
- C would import this as:
 $A \text{ says } p \text{ :- } B \text{ says } q, A \text{ says } r$
- We may represent C's view of A's rule by:
 $A \text{ says } ((B \text{ says } q) \wedge r \Rightarrow p)$
- We may represent C's conclusion by:
 $(B \text{ says } q) \wedge (A \text{ says } r) \Rightarrow (A \text{ says } p)$
- How did we get here?

A logical analysis (cont.)

- So we assume:

$A \text{ says } ((B \text{ says } q) \wedge r \Rightarrow p)$

and would like to derive:

$(B \text{ says } q) \wedge (A \text{ says } r) \Rightarrow (A \text{ says } p)$

- Assume the standard modal axiom

$A \text{ says } (S \Rightarrow T) \Rightarrow (A \text{ says } S) \Rightarrow (A \text{ says } T)$

and the necessitation rule.

- We obtain:

$A \text{ says } ((B \text{ says } q) \wedge r) \Rightarrow (A \text{ says } p)$

and then (only!):

$(A \text{ says } B \text{ says } q) \wedge (A \text{ says } r) \Rightarrow (A \text{ says } p)$

A logical analysis (cont.)

- We can finish with the strong axiom:
 $S \Rightarrow (A \text{ says } S)$
- A weaker form suffices:
 $B \text{ says } S \Rightarrow (A \text{ says } B \text{ says } S)$

Important properties of Binder

- Binder programs can define and use new, application-specific predicates.
- A statement in Binder can be read as a declarative English sentence.
- Queries in Binder are decidable (in PTime).

Questions:

- Should there be more built-in syntax and semantics?
- Can all reasonable policies be expressed?
Can the simple ones be expressed simply enough?
- What about other algorithmic problems?

Data integration

- A classic database problem is how to integrate multiple sources of data.
 - The sources may be heterogeneous. Their contents and structure may be partly unknown.
 - The data may be semi-structured (e.g., XML on the Web).

TSIMMIS and MSL

[Garcia-Molina et al., mid 1990's]

- Wrappers translate between a common language and the native languages of sources.
- Mediators then give integrated views of data from multiple sources.
- The mediators may be written in the Mediator Specification Language (MSL).

```
<cs_person {<name N> <relation R> Rest1 Rest2}>@med :-  
  <person {<name N> <dept `CS'> <relation R> | Rest1}>@whois  
  AND decompose_name(N, LN, FN)  
  AND <R {<first_name FN> <last_name LN> | Rest2}>@cs
```

Similarities

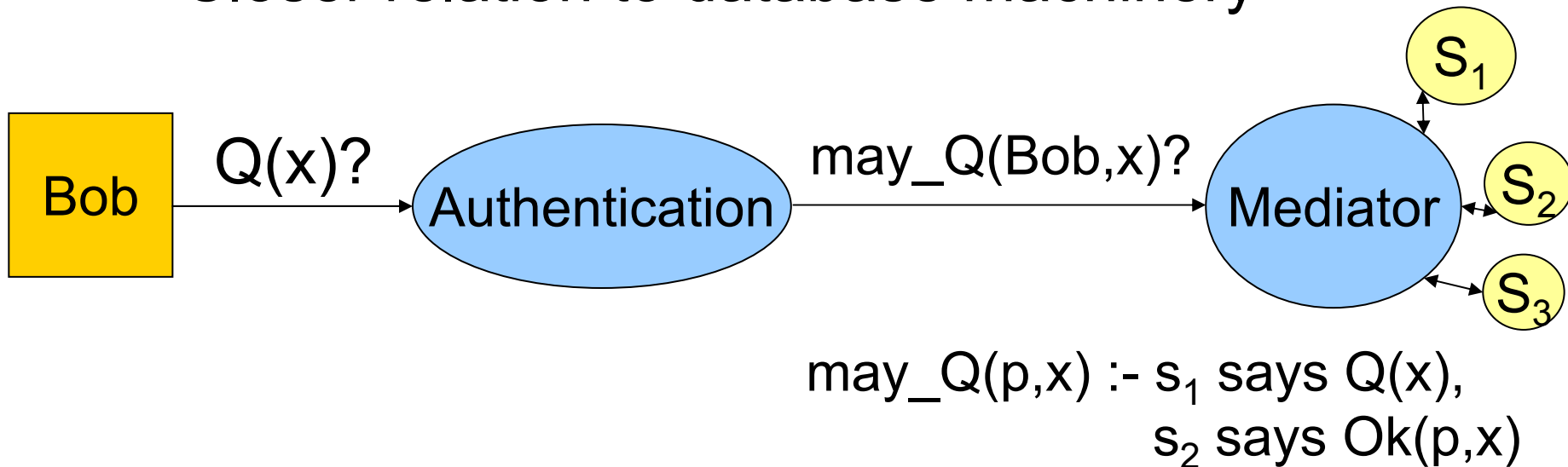
- MSL is remarkably similar to Binder.
 - They start from Datalog.
 - They add sites (or contexts).
 - $X@s$ corresponds to s says X .
 - In $X@s$, the site s may be a variable.
- More broadly, distributed access control is partly about data integration.
 - Binder follows the “global as view” approach (GAV), in which each relation in the mediator schema is defined by a query over the data sources.
 - The converse “local as view” approach (LAV) might not be as meaningful for access control.

Caveats

- MSL and Binder are used in different environments and for different purposes.
 - Work in databases seems to focus on a messy but benign and tolerant world, full of opportunities.
 - Work in security deals with a hostile world and tries to do less.
- Security is primarily a property of systems, not of languages.
Coincidences in languages go only so far.

Potential outcomes (speculation)

- Language-design ideas
 - Constructs beyond Datalog
 - Semi-structured data
- More theory, algorithms, tools
- Closer relation to database machinery



Conclusions and open issues

- Big stakes—apparently getting bigger
- A growing body of sophisticated techniques
- Some art, some science

- Elaborate machinery, but not always easy to explain or to apply reliably
- Substantial, credible efforts, but without full proofs or good metrics of security